Robots and People

Vlad Estivill-Castro

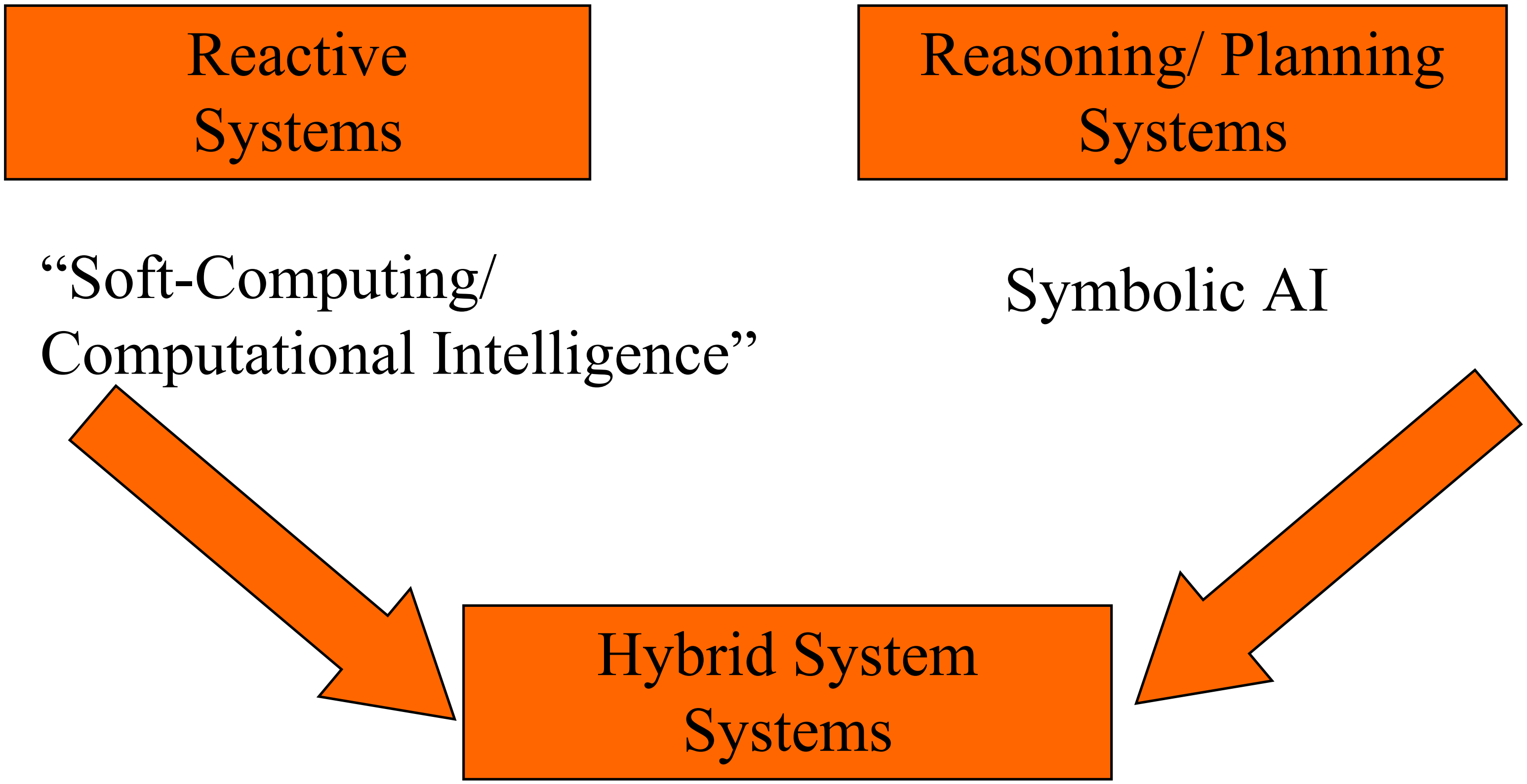
Architecture for Hybrid Robotic Behavior

(D. Billington, R. Hexel and A. Rock)

Software Architecture

Robots and people

- Agents / Robots



How to integrate?

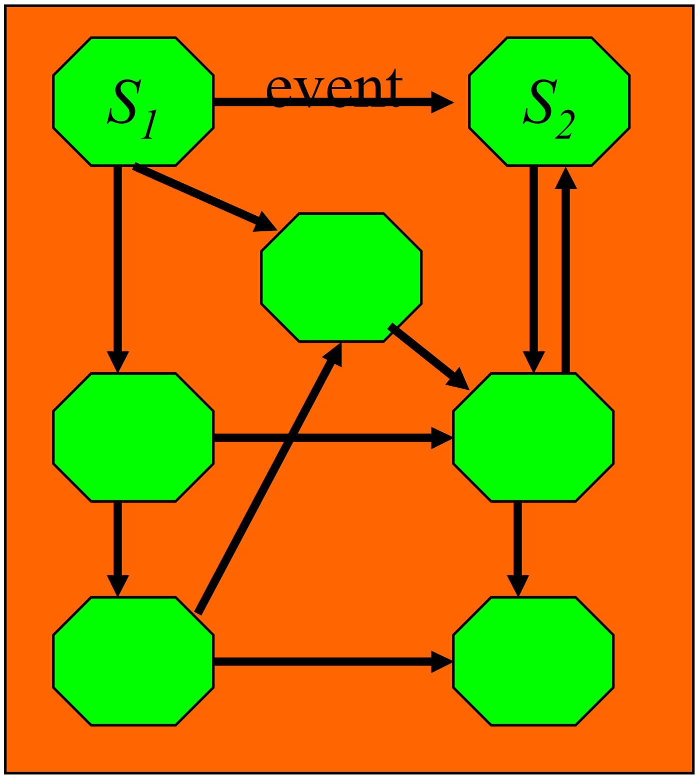
A hybrid system

- The initial progress on logic and reasoning within AI has largely been discarded from mobile robotics in favour of reactive architectures
- We demonstrate the use of non-monotonic reasoning in the challenging application of RoboCup
- Plausible logic is the only non-monotonic logic with an algorithm that detects loops

Robots and People

Hybrid System for Intelligent and Integrated System

- Reactive System
  - State Machine



- Reasoning
  - Non-Monotonic Logic

```
1.  name(Node).
2.  type
    State_Type(S_0,..,S_k).
3.  Ú{State(S_0),…,State(S_
    k)}.
4.  Ú{ØState(S_i),ØState(S_
    j)}. (" i ¹ j)
5.  input{"e_i"}. (for
    i=1,…,k}
6.  Default: Þ State(S_0).
7.  Switch_S_0_S_i:{"e_i"}
    ⇒ State(S_i). (for
    i=1,…,k)
8.  Switch_S_0_S_i >
    Default.
```

Reasoning

- Deriving conclusions from facts
  - Apparently, a fundamental characteristic of intelligence
- An expected aspect of intelligent systems
- Withdrawing conclusions in the light of new evidence is a capability usually referred to as non-monotonic reasoning

Non-Monotonic Reasoning

- **A form of Common Sense**
- **Retract previous conclusions in the light of new evidence**

1. Planes usually leave on time.
2. My flight leaves at 11:00 am.
3. Therefore, I should be at the airport at 9:00am.
4. My flight is cancelled.
5. Makes no sense to take actions for going to the airport early.

Robots and People

Result: Robotic Poker Player

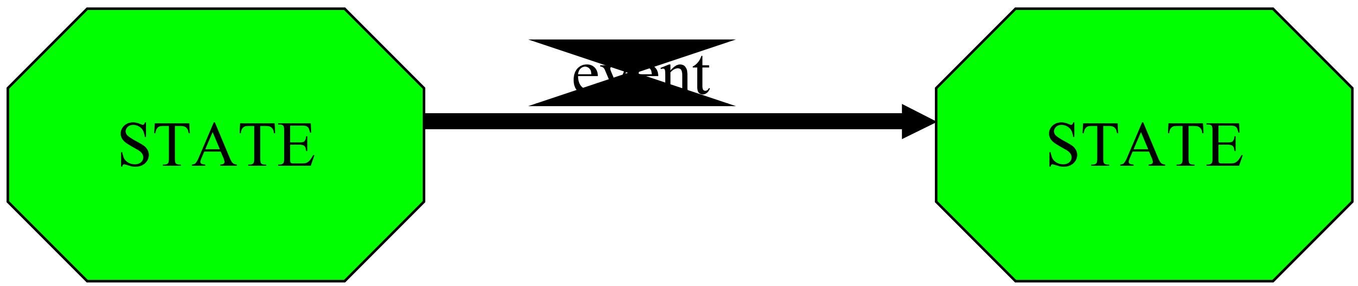
- Integrate
  - Vision
  - Sound recognition
  - Motion Control
  - Reasoning

- Environment
  - Complex
  - Interactive
  - Unpredictable
  - Competitive
  - Incomplete Information

Robots and people

Behaviour Design

- Software Engineering
  - visual models of behaviour



statement from non-monotonic logic

- Behaviour Specification
  - by humans
- Human-Robot Interaction

→ Human-Robot Collaboration

Formal Logics

For the description of the behaviour

Advantages

1. Descriptions are unambiguous
   - Descriptions have specific meanings.
2. Ease of description - descriptive
   - Focus is on what the behaviour does, not how it happens
3. Can be translated to implementations in imperative languages like C++, Java
4. Understandable by humans
   - Can be the result of a knowledge engineering exercise
   - Usually humans describe exceptions and laws governing many situations in this way

Disadvantages

1. Can lead to undecidable settings or other difficulties for implementation, like very large and/or inefficient programs

Previous Work

- Action - Sensor Model [Wooldridge 2002]
  - Solution for control problem
- Golog [Vassos et al 2007]
  - Aim for "Cognitive Robotics"
- Knowledge Middleware [Heintz et al 2007]
  - Bridge low level sensor knowledge
- Robotic Architectures [Liu 2004]
  - Generic Robot [Kim et al 2005]
    - Solution to platform dependence

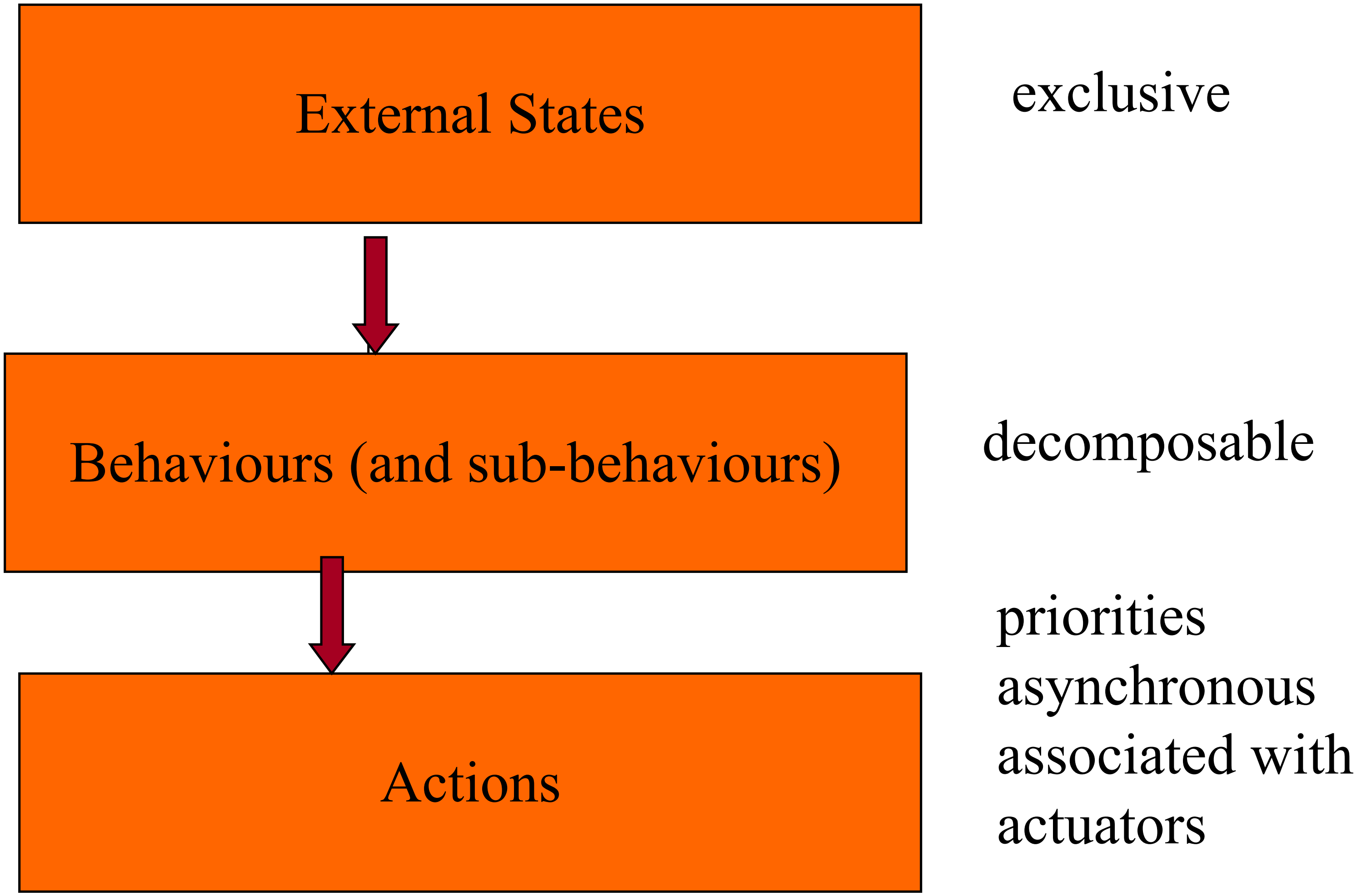
Global Architecture

- Framework = Software Engineering
  - Solves
    - Module Production / Workload problems
    - Software Development Methodology Problem
- Whiteboard (Blackboard [Hayes-Roth 1988])
  - Solves
    - Knowledge representation problem
      - (facts with timestamp and author)
    - Module Interaction Problem
- Domain Knowledge
  - Logics
    - Belief revision / knowledge elicitation
  - Solves
    - Validation / verification /specification

Robots and people

Our Architecture

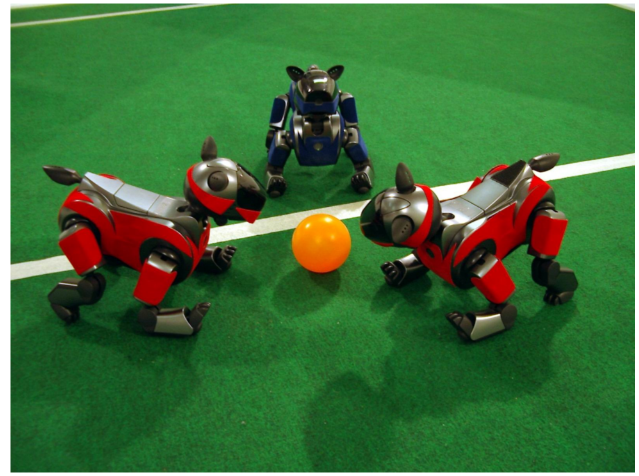
- Solution to Control Problem



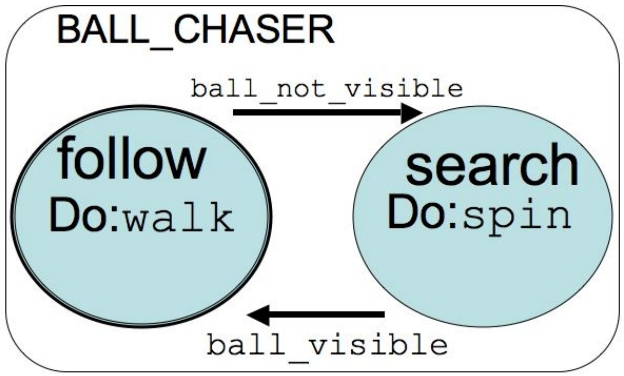
Robots and People

Behaviour Illustration

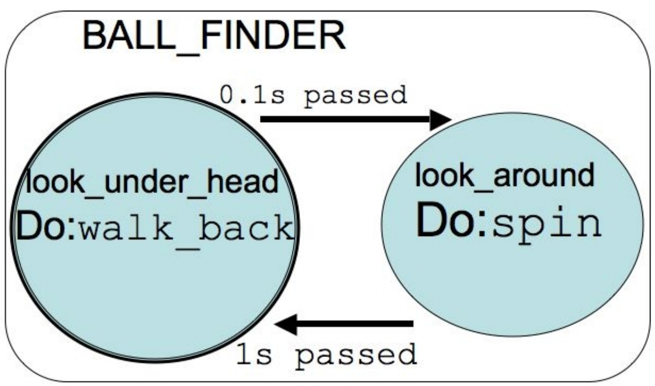
- Robotic Soccer



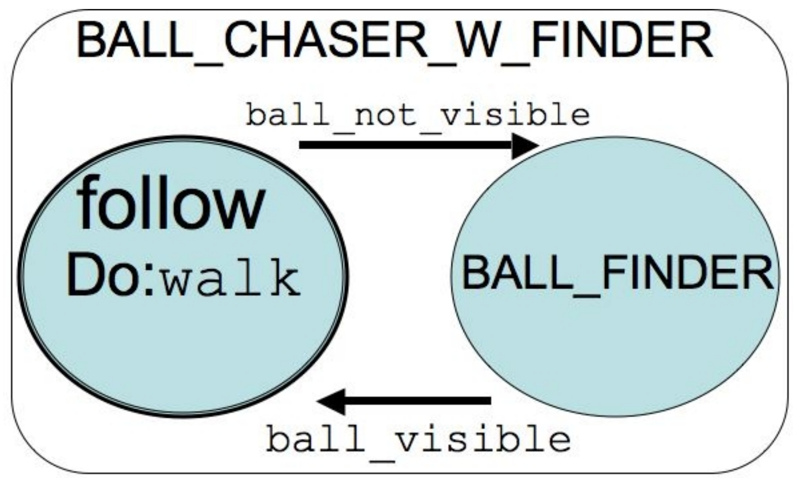
  - Simple Behaviour



  - Sub-behavior



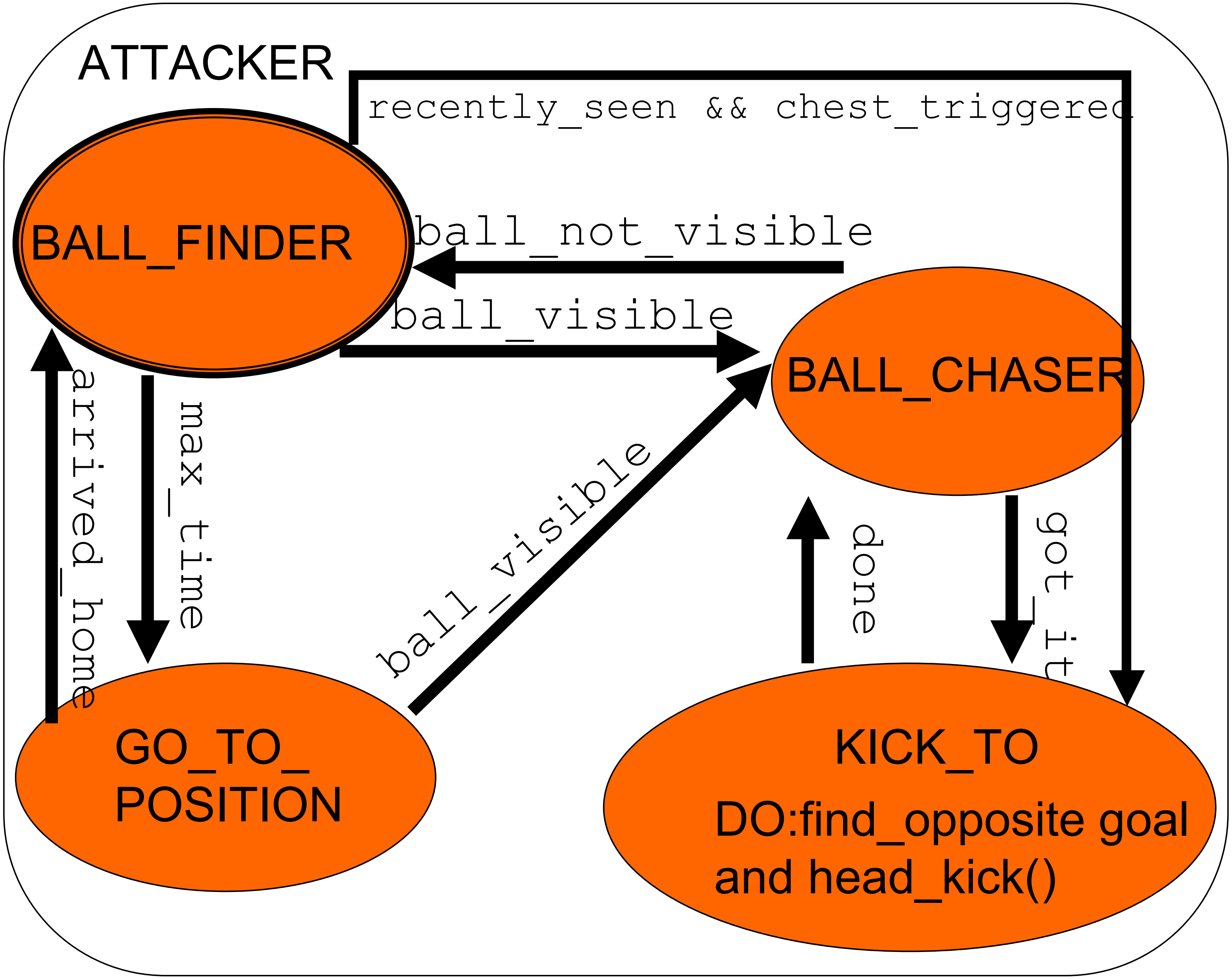
  - Complex behaviour

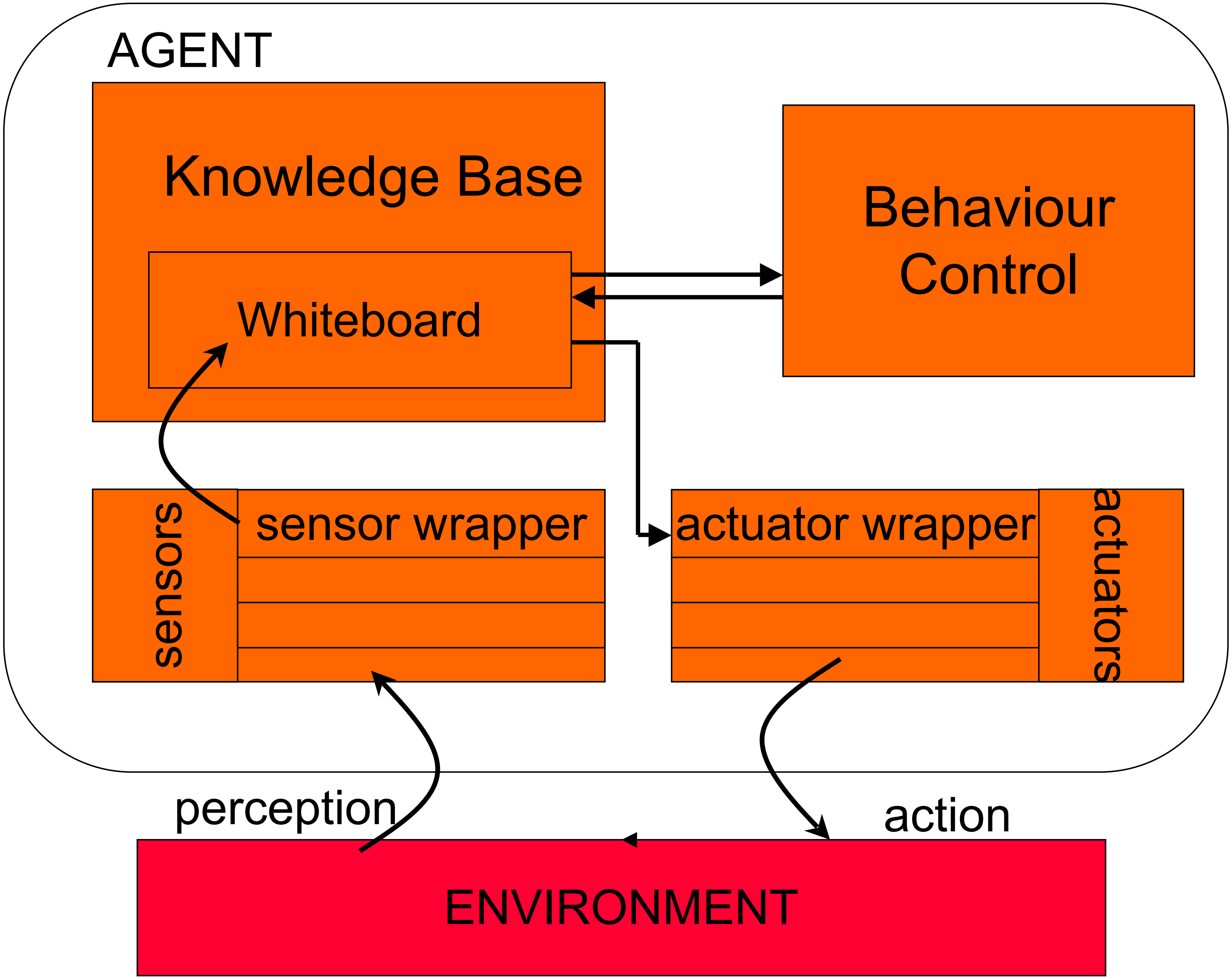


Robots and people

Engineering the behavior

- Using visual descriptions of the behaviour that incorporate formal logic
- Engineers use diagrams to model artefacts.
- Software Engineering has traditionally used diagrams to convey characteristics and descriptions of software

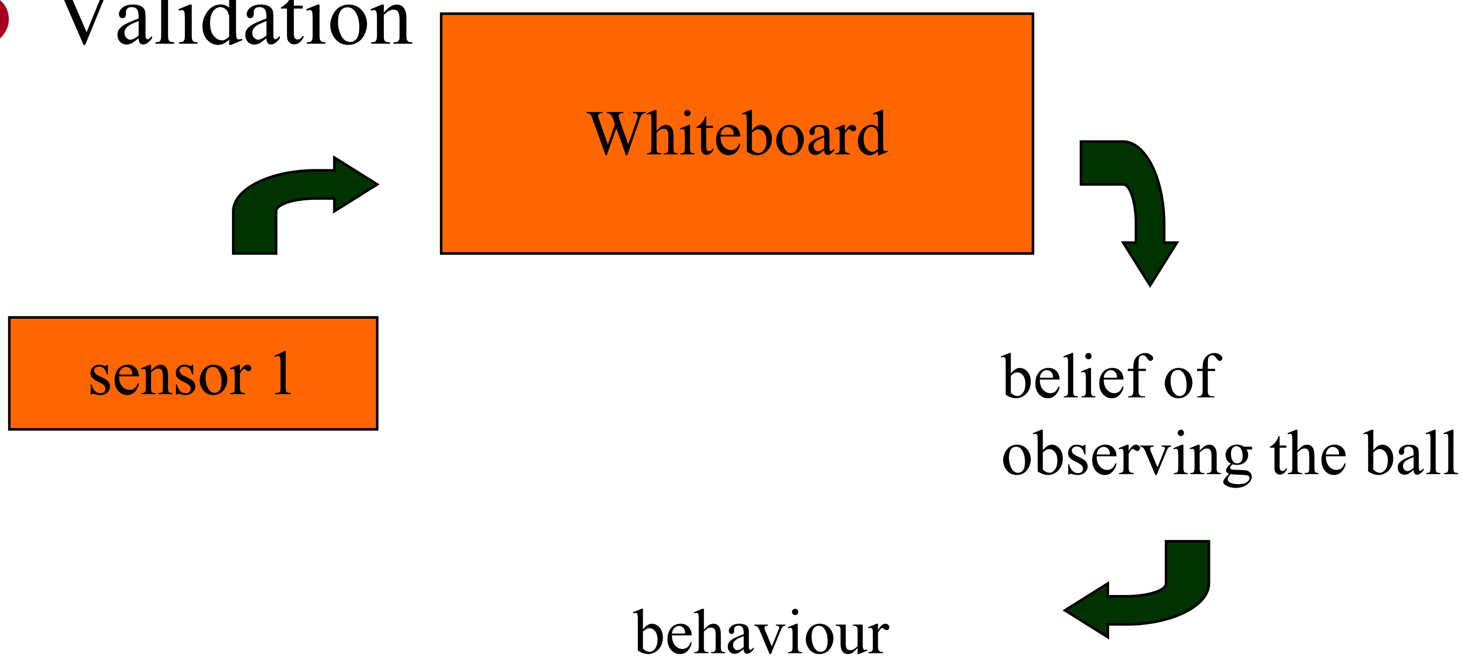

ATTACKER
recently_seen && chest_triggered
BALL_FINDER
ball_not_visible
ball_visible
BALL_CHASER
arrived_home
max_time
ball_visible
done
got_it
GO_TO_
POSITION
KICK_TO
DO:find_opposite goal
and head_kick()


AGENT
Knowledge Base
Whiteboard
Behaviour Control
sensors
sensor wrapper
actuator wrapper
actuators
perception
action
ENVIRONMENT

Robots and People

Wrapping Sensors and Actuators

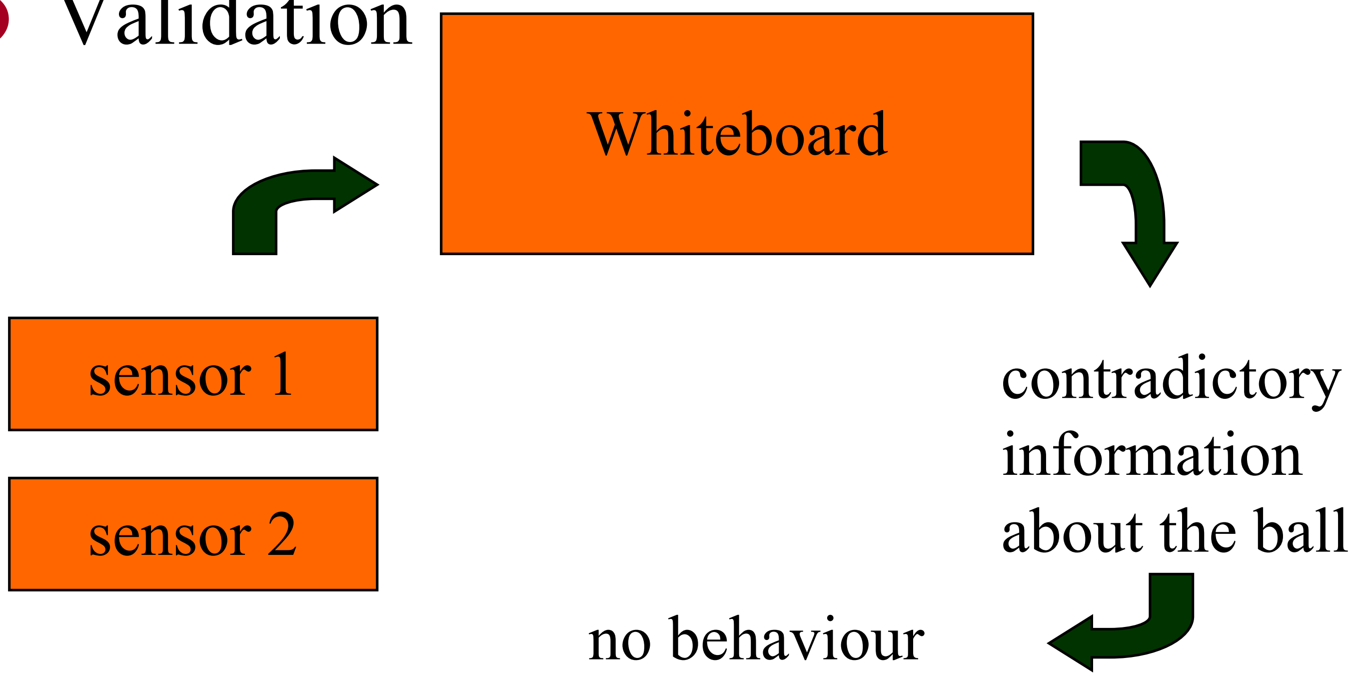
- Portability
- Simulation / Virtualisation
- Validation



Robots and people

Wrapping Sensors and Actuators

- Portability
- Simulation / Virtualisation
- Validation



Alternative
Example: Seeing both goals

Robots and People

Our approach

Vision and Object Recognition

Non-monotonic reasoning

Consistency Module

Sensor fusion

Our approach

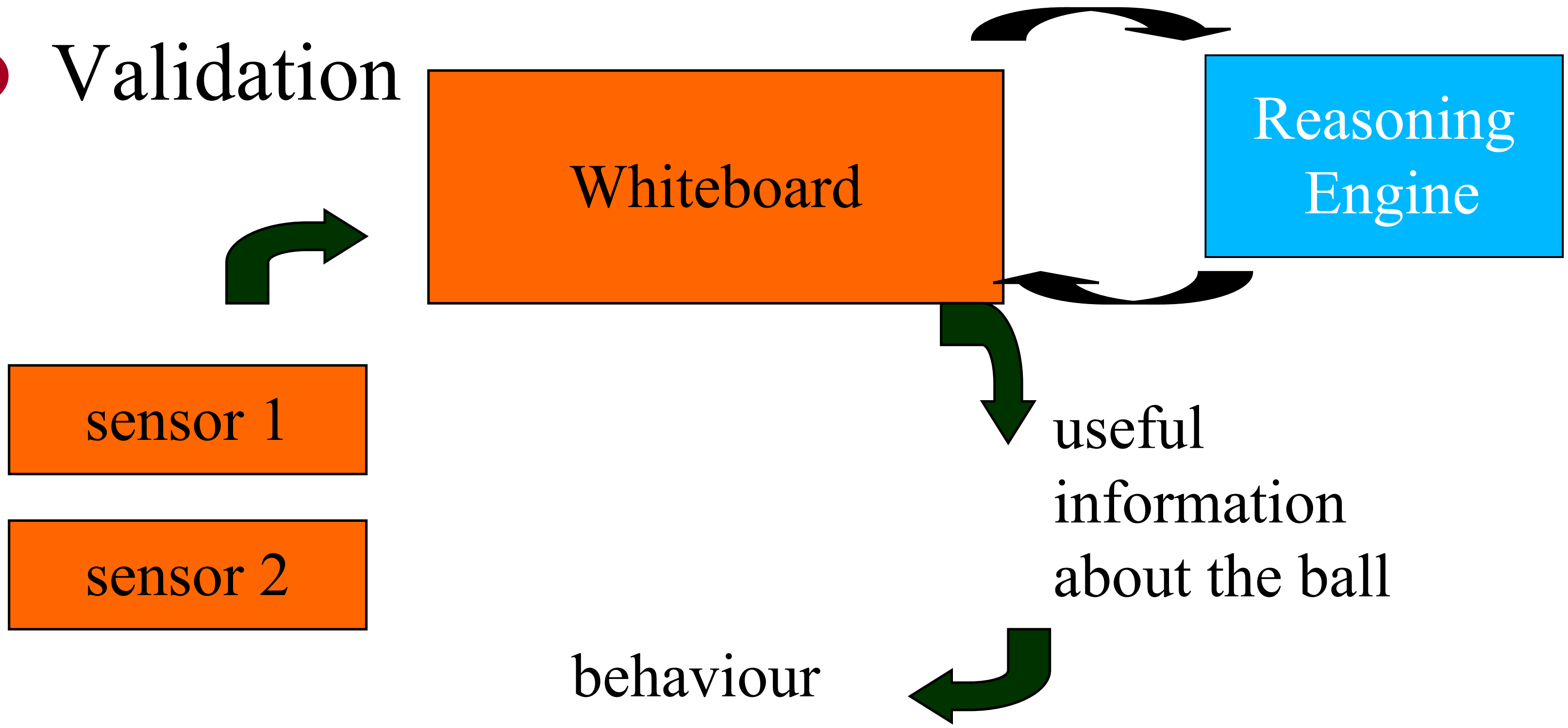
Consistency
Module

Non-monotonic logic that combines facts known about the environment with what is reported by the sensors

Robots and People

Wrapping Sensors and Actuators

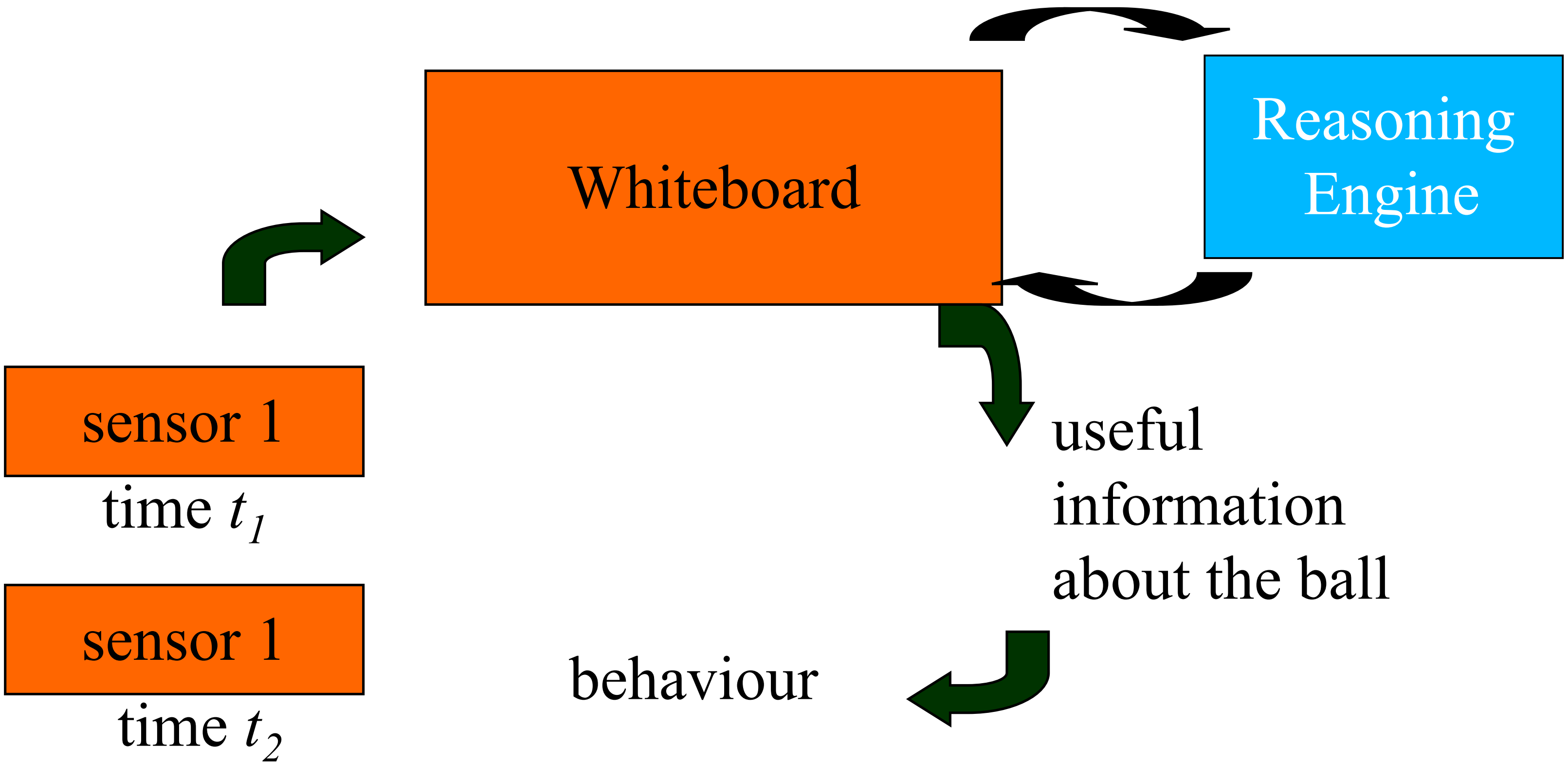
- Portability
- Simulation / Virtualisation
- Validation



Robots and People

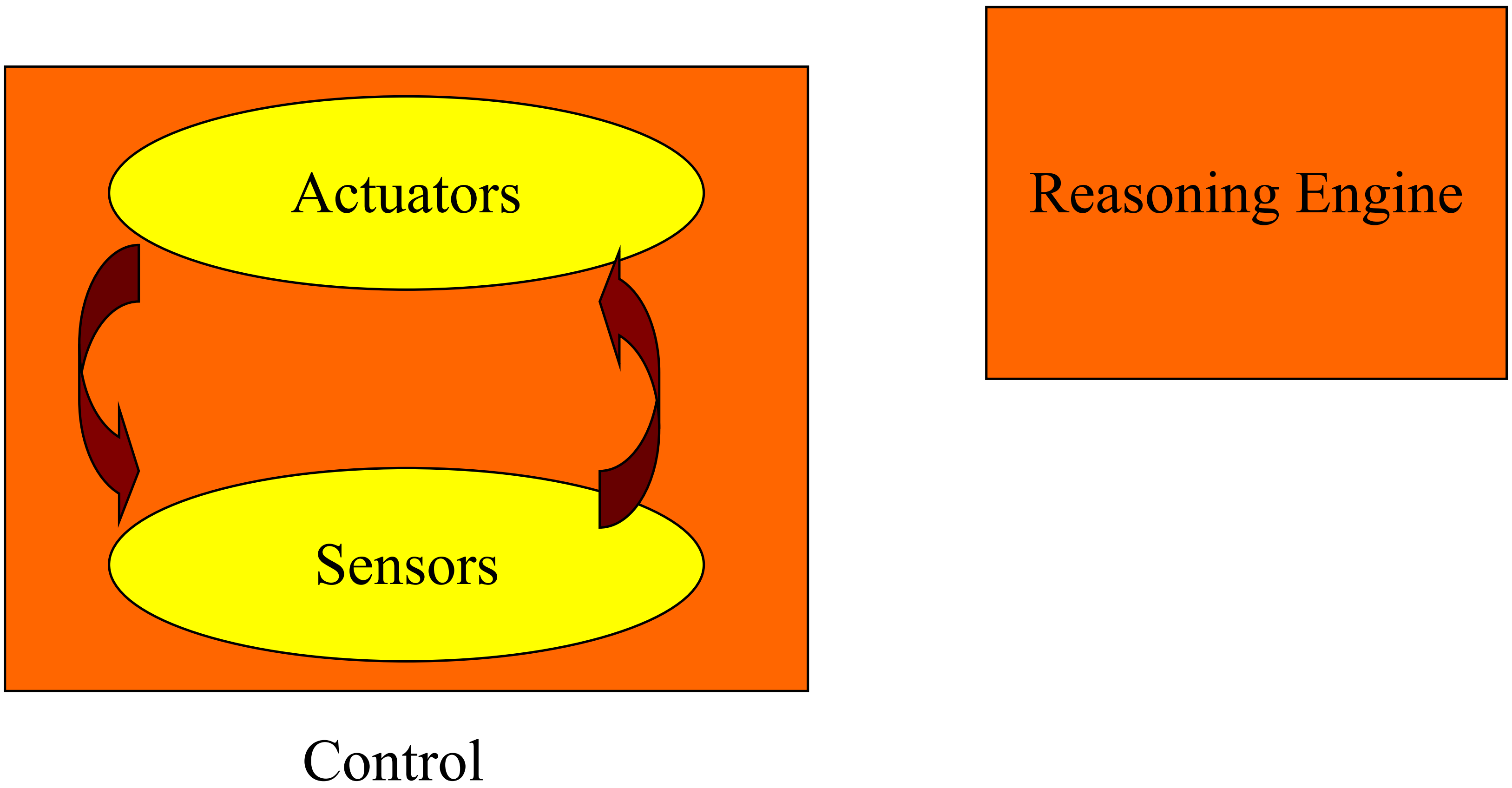
Wrapping Sensors and Actuators

- Fusion in time



Independent and Asynchronous

- Reasoning Engine



Robots and people

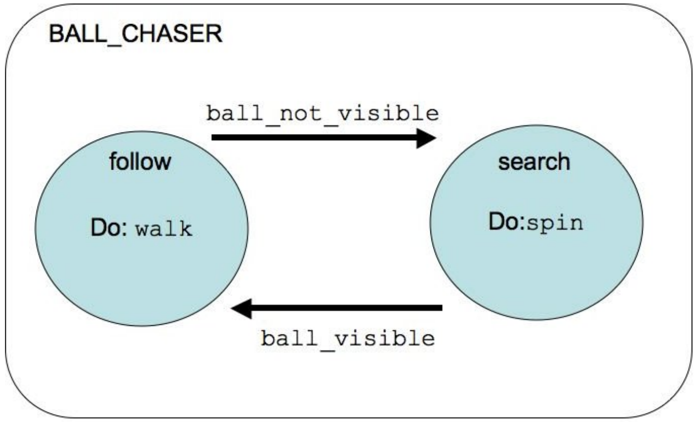
Reasoning Engine

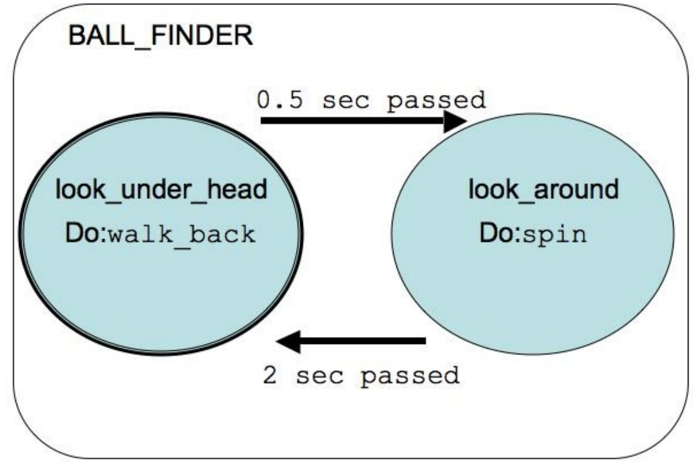
- Template Method

1. New facts are labelled unknown
2. Execute predicates that are more efficient in imperative languages
3. Run the necessary queries /proofs on DPL

Robots and people

Illustration with state diagrams





s_1	$c_1=event_u$	s_i
s_1	$c_2=event_v$	s_j
s_i	$c_t=event_x$	s_p

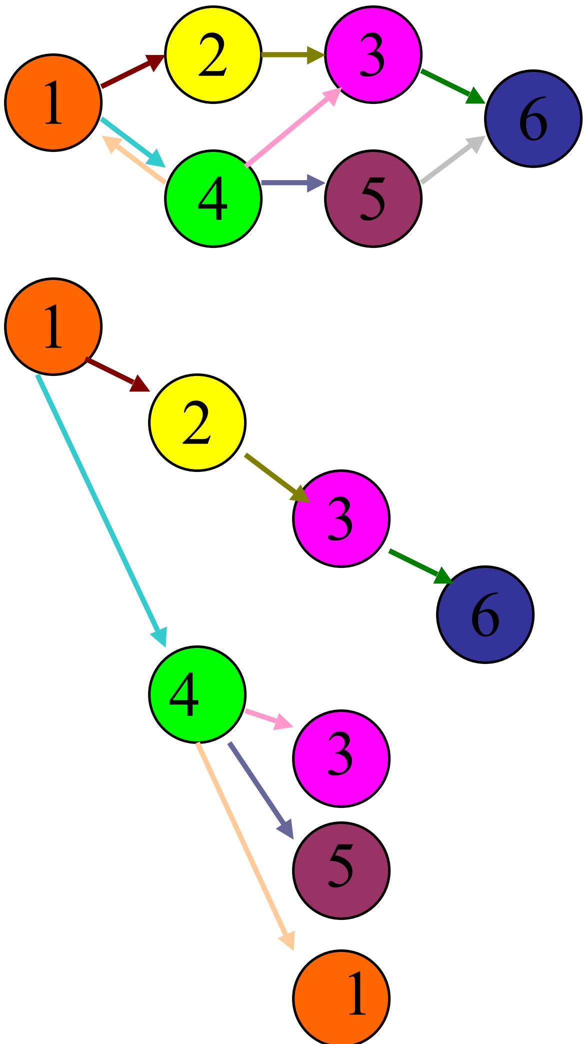
- Exclusivity

$$c_i \wedge c_j = \textbf{false} \; \forall \; i \neq j$$

- Exhaustivity

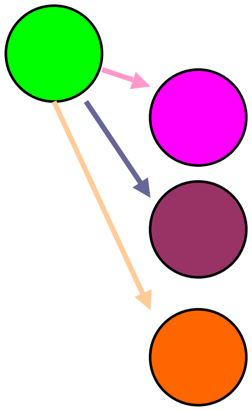
$$\bigvee_{i=1}^{n} c_i = \textbf{true}$$

Convert State Diagram into Behaviour Tree



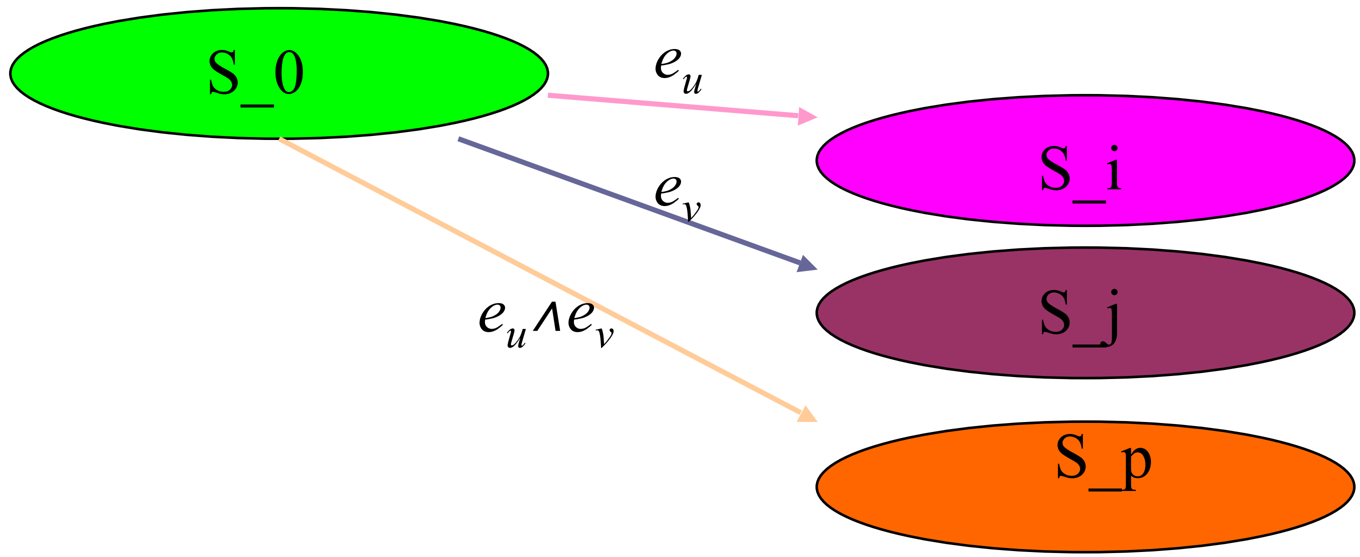
- Draw down by breadth-first search
- Already visited nodes are cloned but not explored again

Convert a node in the tree to a module in Plausible Logic



1. name(Node).
2. type State_Type(S_0,..,S_k).
3. v{State(S_0),…,State(S_k)}.
4. v{¬State(S_i),¬State(S_j)}. (∀ i ≠ j)
5. input{“e_i”}. (for i=1,…,k}
6. Default: ⇒ State(S_0).
7. Switch_S_0_S_i:{“e_i”} ⇒ State(S_i). (for i=1,…,k)
8. Switch_S_0_S_i > Default.

Using the priority relation

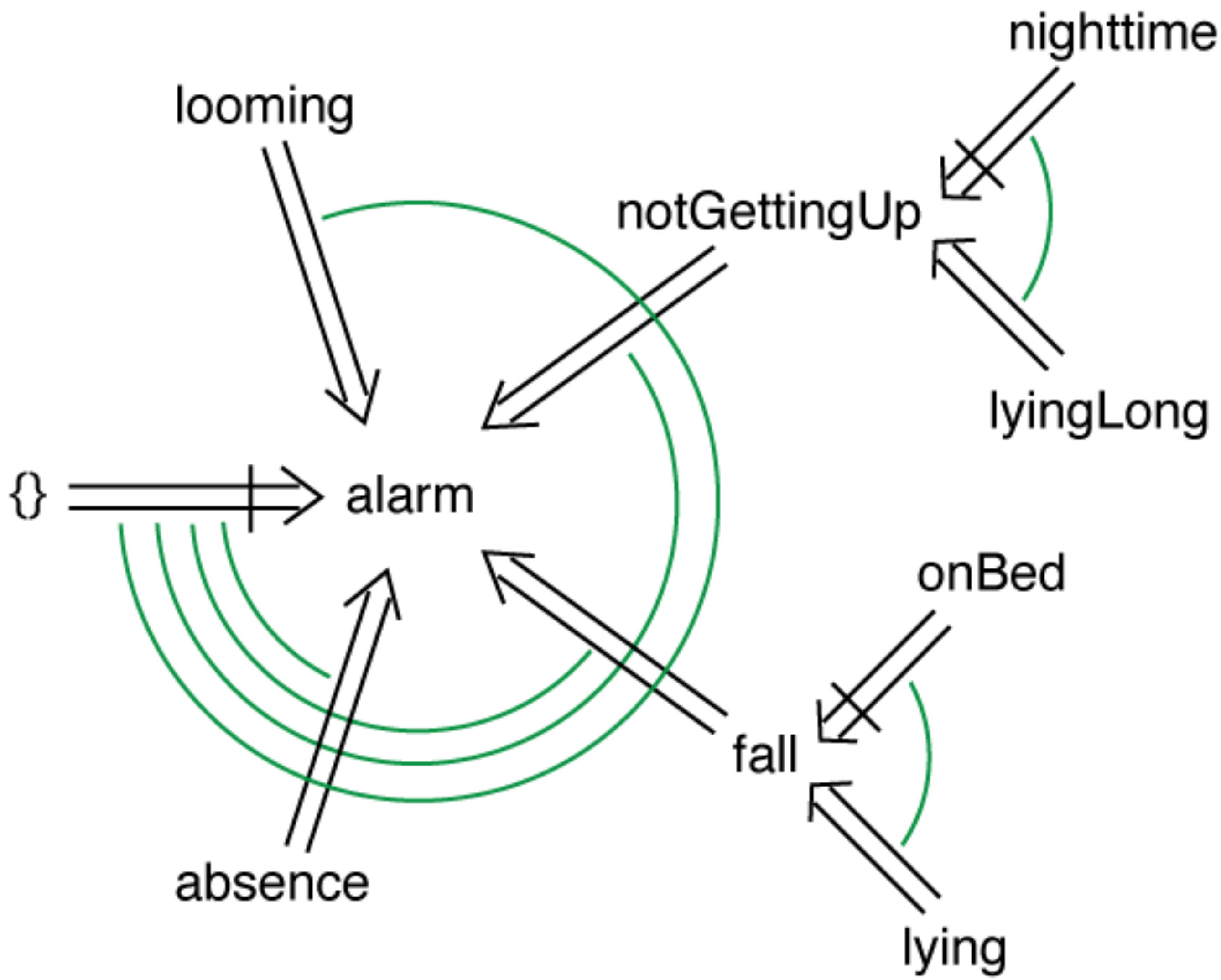


1. Switch_S_0_S_i:{"e_u"} ⇒ State(S_i).
2. Switch_S_0_S_i > Default.
3. Switch_S_0_S_j:{"e_v"} ⇒ State(S_j).
4. Switch_S_0_S_j > Default.
5. Switch_S_0_S_p:{"e_v∧e_u"} ⇒ State(S_p).
6. Switch_S_0_S_p > Default.
7. **Switch_S_0_S_p > Switch_S_0_S_i.**
8. **Switch_S_0_S_p > Switch_S_0_S_i.**

A logic for looking after the lady

1. Usually there is no reason for alarm
2. The absence of owner for a long time is reason for alarm (this takes precedence over rule 1)
3. Lying usually results from a fall
4. A fall is usually a reason for alarm (this takes precedence over rule 1)
5. Being on bed is not a fall (this takes precedence over rule 4)
6. Lying for a long time means owner is not getting up.
7. Not getting up is a reason for alarm (this takes precedence over rule 1)
8. If it is night, it is fine not to get up (this takes precedence over rule 7)
9. If there is a stranger looming over the lady, it is reason for an alarm (takes precedence over rule 1)
10. Owner can't be absent while on bed, or lying or lying for a long time.
11. Owner can't be lying for a long time without lying for a short time.

Diagrams a la Petri nets

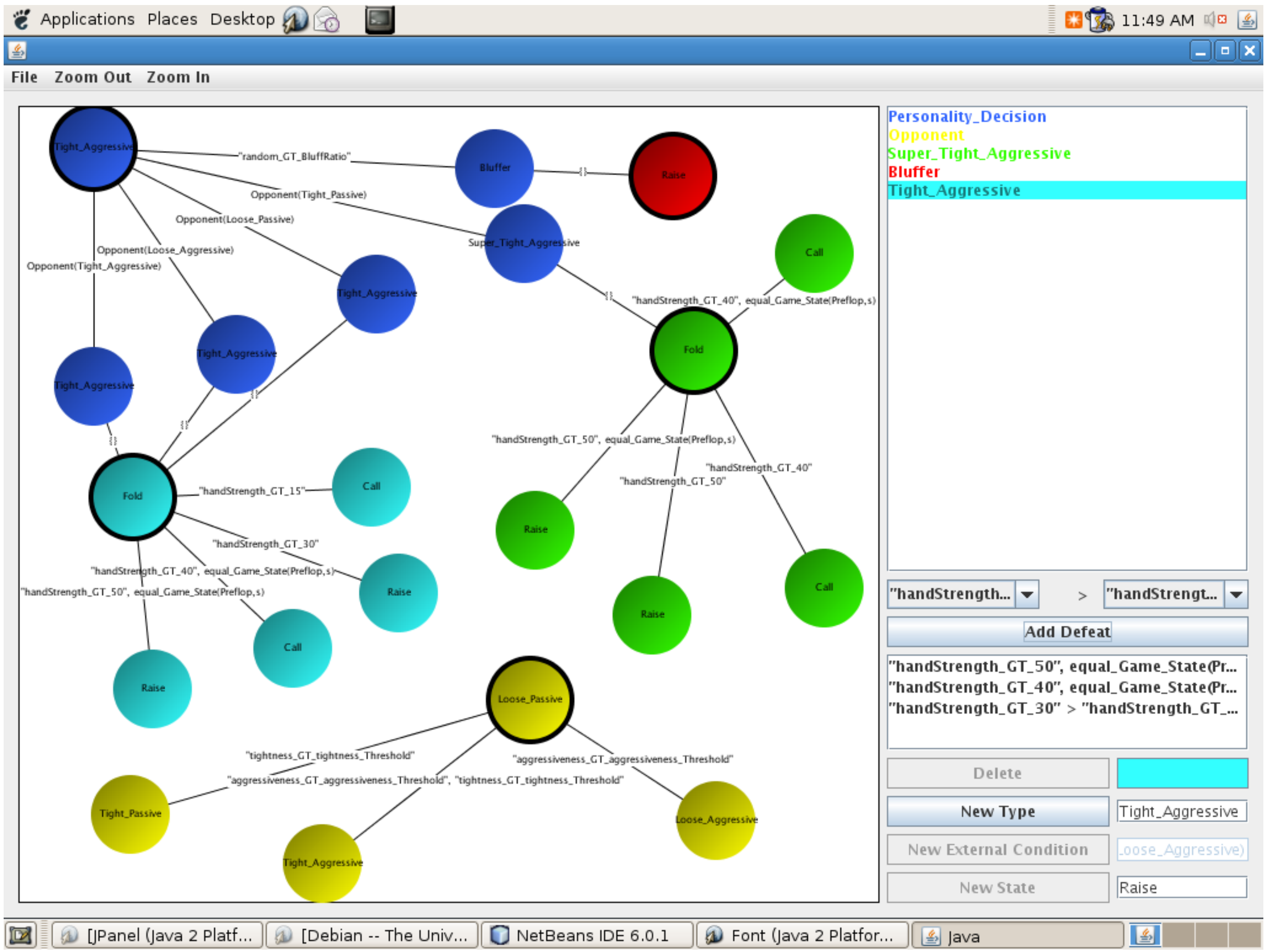


Robots and people

Prototype demonstrated at RoboCup@Home 2007

Robots and People

A diagram for a poker player

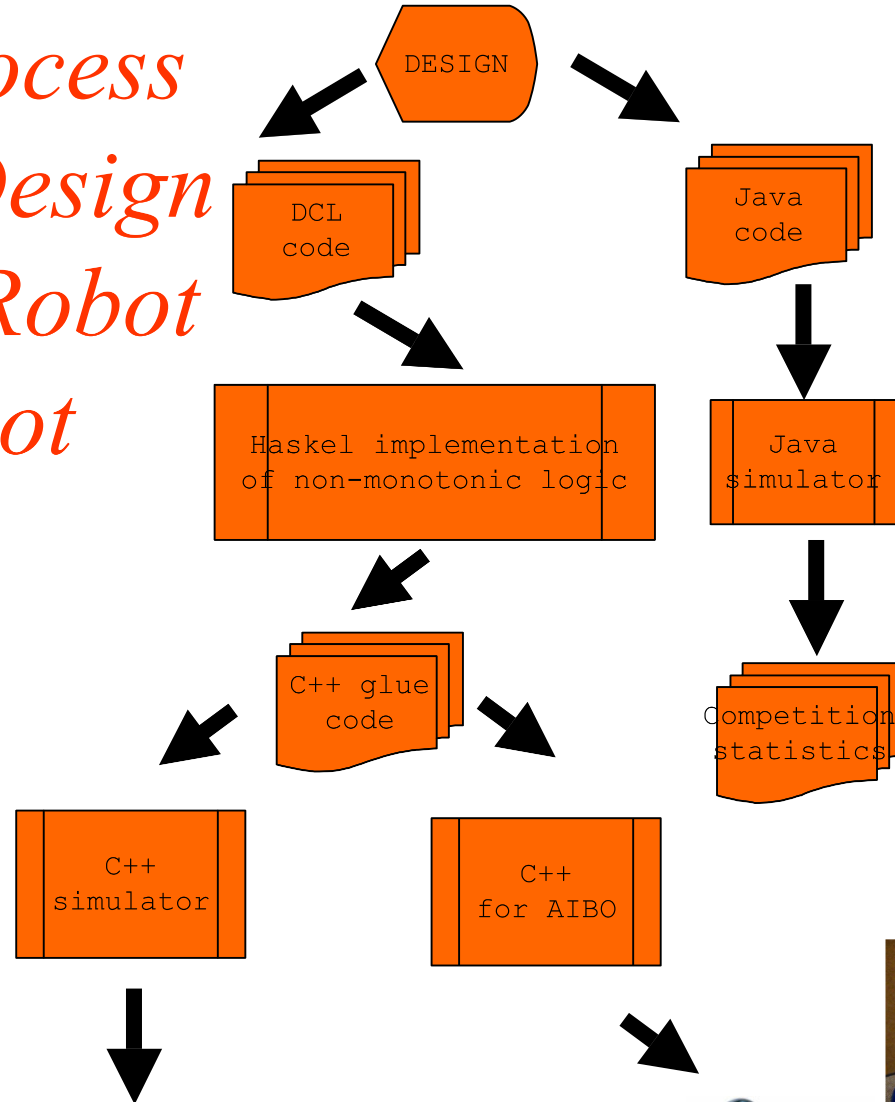


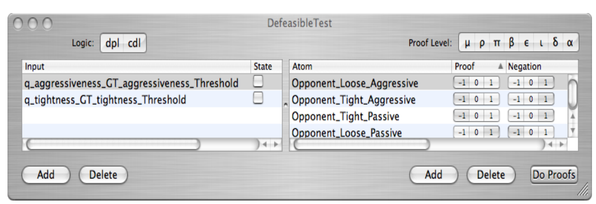
Code generated (example)

```
/* This is code Generated by the DPLGenerator
** This program was made by Mark Johnson 2008 (MiPAL)
** File Opponent.d
*/

name{Opponent}.

type Opponent(x<-Opponent_Type).

type Opponent_Type = {Loose_Passive, Loose_Aggressive, Tight_Passive, Tight_Aggressive}.

\/{Opponent(Loose_Passive), Opponent(Loose_Aggressive), Opponent(Tight_Passive), Opponent(Tight_Aggressive)}.

\/{~Opponent(Loose_Passive),~Opponent(Loose_Aggressive)}.
\/{~Opponent(Loose_Passive),~Opponent(Tight_Passive)}.
\/{~Opponent(Loose_Passive),~Opponent(Tight_Aggressive)}.
\/{~Opponent(Loose_Aggressive),~Opponent(Tight_Passive)}.
\/{~Opponent(Loose_Aggressive),~Opponent(Tight_Aggressive)}.
\/{~Opponent(Tight_Passive),~Opponent(Tight_Aggressive)}.

input{"aggressiveness_GT_aggressiveness_Threshold"}.
input{"tightness_GT_tightness_Threshold"}.

Default_Opponent: {}=>Opponent(Loose_Passive).

Switch_aggressiveness_GT_aggressiveness_Threshold: {"aggressiveness_GT_aggressiveness_Threshold"} => Opponent(Loose_Aggressive).
Switch_aggressiveness_GT_aggressiveness_Threshold > Default_Opponent.

Switch_tightness_GT_tightness_Threshold: {"tightness_GT_tightness_Threshold"} => Opponent(Tight_Passive).
Switch_tightness_GT_tightness_Threshold > Default_Opponent.

Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold: {"aggressiveness_GT_aggressiveness_Threshold",
        "tightness_GT_tightness_Threshold"} => Opponent(Tight_Aggressive).
Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Default_Opponent.

Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Switch_tightness_GT_tightness_Threshold.
Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Switch_aggressiveness_GT_aggressiveness_Threshold.
```

Robots and People

Current Process to Embed Design into AIBO Robot or Nao Robot





Robots and people

Systems interacting with humans

Reasonable Independence of Reasoning Approach

- Forward chaining
  - Start from the current state of the behaviour, run the label of every exiting transition and move to the next state accordingly
  - Illustration
    - Find information about opponent and then decide on the personality to play
      - if opponent is tight and passive, then it is good to adopt an aggressive personality

Reasonable Independence of Reasoning Approach

- Backward chaining
  - Run many of the predicates further down the line, and then be ready to apply and compose them as we move back into the chain of state transitions
  - Illustration
    - Find how would you play (your move) if you were
      - tight aggressive
      - loose aggressive
      - lose passive
      - tight passive
    - consider the opinion of this experts in judging your play in light of the stats you have on your oponent

Modelling behaviours

1. Computer Assisted Software Engineering enables the manipulation of modelling diagrams and the generation of code from the models.
2. We introduce diagrams that use logic to describe behaviour.
3. Our proposal extends techniques like Finite State Machines, Petri Nets Object Models for Object Orientation, and Behavior Trees.
4. We model the relationship between several inputs as asserted conditions about the environment that an agent can reason about (using logics) and resolve with respect to knowledge of the environment.

Summary

- Architecture for behaviors that integrate reactive behavior and reasoned behavior
- Several patterns of software engineering incorporated that enable integration of intelligent capabilities
  - Integrating knowledge representation and control
  - validity / expresibility / platform independence / software process and methodology
- A middleware
  - discussed it mostly OO (modules)
  - but seems possible to integrate agents
    - illustration of asynchronous achievement of goals by backward / forward chaining

Robots and people

THANK YOU